ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUDPL01

REPRODUCE ANYTHING, ANYWHERE: A GENERIC SIMULATION SUITE FOR TANGO CONTROL SYSTEMS*

S.Rubio-Manrique, S.Blanch-Torné, M.Broseta, G.Cuní, D.Fernández-Carreiras, J.Moldes, ALBA-CELLS Synchrotron, Barcelona, Spain
Andrew Götz, ESRF, Grenoble, France

Abstract

Synchrotron Light Sources are required to operate on 24/7 schedules, while at the same time must be continuously upgraded to cover scientists needs of improving its efficiency and performance. These operation conditions impose rigid calendars to control system engineers, reducing to few hours per month the maintenance and testing time available. The *SimulatorDS* project has been developed to cope with these restrictions and enable test-driven development, replicating in a virtual environment the conditions in which a piece of software has to be developed or debugged. This software provides devices and scripts to easily duplicate or prototype the structure and behaviour of any *Tango Control System*, using the *fandango* python library to export the control system status and create simulated devices dynamically. This paper will also present first tests using multiple SimulatorDS instances running on a commercial cloud.

INTRODUCTION

The classical paradigm for building a simulation infrastructure is the development and prototyping of a new control system, a period in which a software team is already committed to the development of a complex system but doesn't have yet access to the real hardware nor infrastructural resources[1].

Software Testing Techniques

Continuous Delivery and other widespread methodologies emphasize the importance of delivering well-tested systems to operators, as it will build confidence over the delivered system[2]. The two most common techniques for software testing are *Unit Testing* and *Functional Testing*.

Unit testing is a well established technique in the software field for validating libraries and API's; based on breaking up library components in basic software procedures that can be validated against well specified interfaces. It's widespread and well-established, but costly to apply in already existing or inherited projects where documentation may be scarce or outdated.

Functional testing instead proceeds to test the different parts of the application behaviour, validating that each of the functionalities of the program behaves according to the expected results. Depending of the scope and objective of the tests it can be known as integration testing, regression testing, usability testing, smoke testing, sanity testing amongst others.

* Work supported by the Tango Consortium and the European Synchrotron Radiation Facility

PySignalSimulator

In the case of most control systems, it is not possible to do a functional test without a certain capability to simulate the hardware system to be controlled.

The *PySignalSimulator* Tango Device Server was developed to cover these needs during ALBA Synchrotron construction phase. It provided an easy-to-configure[3] generic simulation tool for testing graphical applications. Based on single-line formula definitions stored in the Tango database (Table 1), it allowed to simulate hardware and to test the control system infrastructural services, like archiving[4] or alarms[5]. The success of this approach enabled other Tango Collaboration[6] members like MAX IV to reuse our developments during its design and development phases.

Table 1: Dynamic Attributes as Declared in Tango DB

```
Square=0.5+square(t)  #(t = seconds since the device started)
NoisySinus=2+1.5*sin(3*t)-0.5*random()
SomeNumbers=DevVarLongArray([1000.*i for i in range(1,10)])
```

But, once ALBA Synchrotron entered in operation, new necessities appeared for simulators that a simple approach like PySignalSimulator was not able to cover. For a facility in a growing phase like ours, upgrades of the control system are required monthly, in systems as critical like Linac injection modes, Radiofrequency upgrades, orbit feedback improvements, … Upgrades that must be applied without interrupting the current operation schedule of 5912 yearly hours of beam for users, a constrain that limits the availability of hardware for testing to a few hours per month.

This lack of testing time availability created the need of an automated way of validating updates by replicating a running Tango Control System in a test environment.

SIMULATORDS

Several studies [7] has been done on the advantages of using simulation environments for software development. These works have pointed out that the effort of building a fully detailed model is often hardly justifiable if the work needed to implement a simulation environment must doesn't tend towards 0.

This lack of profitability of simulation environments is often caused by the impossibility to replicate the real hardware infrastructure or the obsolescence of the simulation design, as real systems tend to change a lot during building phases.

The SimulatorDS [8] package has been developed to overcome these certain limitations of the model-based

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUDPL01

approach. It evolved from PySignalSimulator device server to provide a data-driven approach, easier replication and a generic framework for testing.

In order to evolve our existing simulation tools, several needs has been identified to be covered by the simulation framework:

- Control System Replication: create a simulated copy of a running system.
- Continuous Integration: validate effect of software changes during development.
- Functional testing: validate the behaviour of a whole control system before a part of it is upgraded.
- Integration tests: validate external software upgrades (OS, libraries, tools) against our own software.
- HMI Validation: Provide a testing platform for GUI's whenever real hardware availability is scarce.
- Chaos engineering / Canary testing, prepare test cases and validate them under production conditions.
- Proof of concepts, test new technologies.
- Bug reproducibilty: recreate bug conditions to be included in issue reports for external teams.

The required functionalities to achieve these objectives have been split in two different pieces of software: the *SimulatorDS* Tango Device Server for executing the simulation itself and the *gen_simulation* script for configuring the testing environment and facilitate the generation building.

Both developments are focused on not starting new tools from scratch but on exploiting the already existing features of existing python libraries in Tango: *PyTango* and *fandango*.

All the features of SimulatorDS are possible due to the versatility of Tango Python binding, *PyTango[9]*. It allows the flexibility of a dynamic typed language and mutable objects, providing device servers that can be expanded on-the-run with new attributes and behaviours.

The SimulatorDS Tango Device Server

As an evolution of the PySignalSimulator device server, the SimulatorDS device is a wrapper around the DynamicDS class of the *fandango* python library [10]. This device server reads single-line python code [recipes] from the Tango Database for each declared attribute, command, quality or state.

The device extends the python built-ins with mathematical and statistical methods. Other methods available[11] include time conversion, external attribute reading, database access (Fig.1), regular expressions. Additional libraries and declarations can be loaded using ExtraModules and LoadFromFile properties (Fig. 2).

```
CRef=float(HDB.get_attribute_values('sr/di/dcct/current',str2time('2017-03-01'),'+300')[N])
PRef=float(HDB.get_attribute_values('sr/vc/all/pressure',str2time('2017-03-01'),'+300')[N])
Noise=sin(PI*t)
Pressure=PRef+1e-7*Noise
Current=CRef+2*Noise
```

Figure 1: DynamicAttributes declaration accessing archived data via ExtraModules.

Device properties [test/sr/di]

Property name	Value
CheckDependencies	True
DynamicAttributes	CRef=float(HDB.get_attribute_values('sr/di/dcct/current',str2time('2017-03-01'),'+300')[N]) PRef=float(HDB.get_attribute_values('sr/vc/all/pressure',str2time('2017-03-01'),'+300')[N]) Noise=sin(PI*t) Pressure=PRef+1e-7*Noise Current=CRef+2*Noise
DynamicStates	ALARM=Pressure>1e-8 MOVING=0<Current<20 ON=Current>=20 OFF=1
ExtraModules	PyTangoArchiving.Reader as HDB lifetime_lib
KeepAttributes	True
KeepTime	500
LoadFromFile	/control/sr_di_calc.py
LogLevel	WARNING
UseEvents	false

Figure 2: Device Properties of a SimulatorDS, as seen in Jive configurator tool.

The evaluation engine permits to codify complex state machines through the VAR and PROPERTY methods, that allows to pass/store/retrieve between dynamic code evaluations and keep the current state of the device, either in memory (using VAR(name,value)) or persistently in the database (WPROPERTY(name,value)).

```
#############################################################################
#                          SIMULATION                                       #
#############################################################################

#                        Constant Parameters                                #
#############################################################################
Uo = 1
Uc = 1.1
Rs = 3.3e6
Qo = 29500
PI = 3.1415927
Working_Frequency = 499650000
B06A = 2.701
B06B = 2.687
B10A = 2.4582
B10B = 2.5467
B14A = 2.678
B14B = 2.5639

#                      Select ID open or ID closed                          #
#############################################################################
# Flag to control if the IDs are open: True when open; False when closed
idsOpen = bool(VAR('idsOpen',VALUE) if WRITE else VAR('idsOpen') or False)
U = Uo if idsOpen else Uc

#                          Select Ibeam                                     #
#############################################################################
machinelbeam = XATTR('sr/di/dcct/averagecurrent')
usrDeflbeam = float(VAR('UserCurrent',VALUE) if WRITE else VAR('UserCurrent',default=130))
setUsrDeflbeam = bool(VAR('usrIbeam',VALUE) if WRITE else VAR('usrIbeam', default=False))
Ibeam = usrDeflbeam if setUsrDeflbeam else machinelbeam
I = Ibeam/1000
```

Figure 3: User defined specification of attributes and constants for simulation, using XATTR to acquire real data from devices while simulating the rest.

Via ExtraModules and LoadFromFile properties, the expert user can load its own constants, formulas, classes and objects and export them to the evaluation engine. Objects declared this way can be used to keep the internal StateMachine model of the simulated device.

States of SimulatorDS devices are declared on a list (Fig.5) and evaluated sequentially, applying the first state which formula evaluates to True.

DynamicStates	ALARM=Pressure>1e-8 MOVING=0<Current<20 ON=Current>=20 OFF=1

Figure 5: DynamicStates declaration.

As seen in Fig. 6, there are several properties that allow to configure the internal evaluation process from Jive. KeepAttributes, KeepTime and CheckDependencies will

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUDPL01

control for how long formulas result are kept instead of being re-evaluated, as well if its results have to be kept at all. These properties will manage both cpu usage and memory usage.

Device properties [test/sr/di]

Property name	Value
CheckDependencies	True
ExtraModules	PyTangoArchiving.Reader as HDB lifetime_lib
IgnoreList	
KeepAttributes	True
KeepTime	500
LoadFromFile	/control/sr_di_calc.py
LogLevel	WARNING
SortLists	True
UseEvents	false
UseTaurus	False

Figure 6: Properties to configure the internal evaluation engine of SimulatorDS.

Copy and Paste: the gen_simulation Script

One of the main objectives of the SimulatorDS project was to simplify the replication of existing systems to be able to debug existing bugs without no need of a real hardware or infrastructure behind.

The process of replicating a running system has been divided in multiple steps:

- Extract all attributes accessed from a PyTango application.
- Export all its values and data configuration to a data file (csv, json or pickle).
- Generate a SimulatorDS default property specification for each Tango class that has been exported.
- Recreate the exported data as new simulators in a different tango-host, mimicking the original structure but with entirely simulated data.
- Apply or modify the events/polling configuration data of all the exported devices to reproduce the original conditions or experiment with new ones.

These tasks are fulfilled by the *gen_simulation* script, a guided helper that allows to perform each step sequentially generating intermediate files that can be modified during the process. It allows to modify selectively the device names or its properties during the replication process.

A Tool for Bug Reproducibility

Bug unreproducibility is a common issue when sharing open source projects amongst several institutes. As we rely on several open source packages to operate our machine (Tango, Taurus, PyTango, HDB++) sometimes we find bugs that are not easy to reproduce at all by the developer teams of these projects in other sites.

A simulation platform that imports/exports files that can be shared and launched anywhere helps to provide reproducibility of bugs encountered in the system so developers outside of the institution can work on them without access to the real system.

Simulating the Control Infrastructure

In addition to simulators, many other devices play a role on simulating the whole control system.

Some of these devices involved are:

- Tango Starter devices (daemon-like processes managing running Tango servers on each host) allow to easily restart device servers with adding new devices to them or modifying its interfaces.
- ProcessProfiler Device Server, it obtains CPU and memory statistics for all processes running in the machine and converts it into Tango attributes, that can be used for archiving or alarms.
- HDB++ Archiving Database [hdbpp], to log and register the whole simulation process, allowing to generate statistics afterwards.
- WorkerDS: A device capable of running maintenance scripts (cron-like), either to clean up the archiving database or execute system maintenance tasks.
- PyAlarm Device Server, the PANIC Alarm System device [12], that may trigger alerts in case of abnormal behaviour of the simulation.

The HDB++ database [13] is a perfect tool for storing data results, as it subscribes to the events pushed by the simulators in a non-intrusive way.

The CSV / JSON files used by gen_simulation script can also be used to store / recreate all these infrastructural devices, or to configure new ones depending on the needs of our tests. The CSV format provides a human-readable way to declare new test cases and setup the host/server/device distribution. These approach has been used to configure the PANIC Alarm System test suite.

OTHER SIMULATION TOOLS IN TANGO

Although SimulatorDS allows final-users to place their simulation code in Python files that will be imported by the device, this kind of implementation doesn't provide any additional feature respect of using the high-level API of PyTango, that allows to define Tango device servers with few lines of code.

The power and versatility of SimulatorDS resides instead on exploiting the Tango Database as repository of rules for the simulation. Using the database allows to freely move devices between machines or update the behaviour with no need of restarting the processes. That's the most prominent feature of SimulatorDS in comparison with the python high-level API or tango-simlib library. Any code loaded at startup can be modified at runtime to provide different behaviour if needed.

The tango-simlib library [14] has been also recently developed for the large telescope array SKA [15]. Using a combination of model-based and data-driven simulation techniques it provides similar features to SimulatorDS. As both tools may be complementary we are actually including it in our Tango 9 tools benchmark setup.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUDPL01

TESTING IN PRODUCTION

Continuous Delivery at ALBA

A technique for validating software in production is Canary Testing, implemented at ALBA Accelerators System for all control packages since 2014. New versions of control software are not deployed for the whole accelerator system but to a well delimited part of it. It allows for early detections of problems while limiting the impact to users.

Chaos Engineering

Chaos engineering [16] is the discipline on running controlled experiments on production systems in observer its behaviour. It increases the knowledge about the limits and weaknesses of a system and increases our level of confidence.

It is based on five principles[17]:

- Build hypothesis around steady state behaviours.
- Vary real-world events.
- Run experiments in production.
- Automate experiments to run continuously.
- Minimize blast radious.

Chaos engineering is starting to be applied at ALBA for doing specific experiments on the performance of the Control System. ALBA is actually migrating its control system from a polling-based approach in Tango 7 to an event-based approach using Tango 9 and ZMQ; in which events can be triggered either on internal polling or asynchronous pushing.

All these variants affect in different way the behaviour of devices, increasing their cpu or memory usage due to the internal threading and cacheing. It also may affect the behaviour of commands, as an slow implementation of a command may trigger unexpected timeouts on clients.

In the case of a layered system, problems are often not caused by any concrete element of the system but by the aggregation of them (accumulated delays, memory leaks, unexpected logics, lost messages that lead to wrong states). Thus makes important to have some ways to "degrade" system conditions in the simulation to study the reliability and fault-tolerance of the system.

Machine runs are actually being used to experiment with the different event configurations available in Tango, trying to get the optimal configuration for each device class.

Testing Tango on Amazon Web Services

In order to perform full integration tests of new releases of tools in Tango, a platform for testing and Chaos Engineering is being developed using Amazon Web Services (AWS) and its Elastic Computing Cloud (EC2).

The objective of the testing is having a full control system infrastructure to validate the effect of a change or upgrade on any part of the system, doing both integration tests for new features or regression tests to discard loses of performance.

A platform for integration testing must allow to run as much layers of the real control system as possible, thus all devices involved in testing (HDB++, WorkerDS, ProcessProfiler, Starter) must be available.

As shown in Fig. 7, a central node is deployed as database for the test, and then successive EC2 instances are launched via aws-cli tool. Each of the nodes reuses an already created image with all tango services.

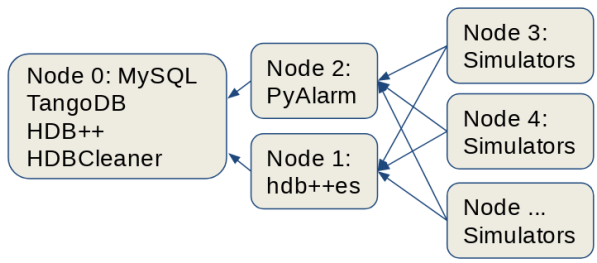

Figure 7: Nodes created in EC2 for each test.

Using the CSV specification of hosts, devices and servers; the fandango library and its Astor API is used to create and configure each device and start them on each node using the desired distribution for the test. Events, polling and archiving configuration is done in the same CSV files for each test case.

CONCLUSIONS

Classical testing, like unit or functional test, have limited usage if they validate the software in conditions that do not match those occurring in production. Checking code correctness is a must, but not enough as many other factors may affect the behaviour of software when it is deployed in real production (problems of scale, delays and timeouts, memory leaks appear, conflicts between tools, unwanted regressions, ...).

To detect those problems before the software gets deployed in the whole system we can apply two other techniques: Canary testing and Chaos engineering. But, with the current tight schedules, there's very limited time to do these experiments. We must arrive to the production stage with our software ready to be validated and with the tools in place to detect any abnormal behaviour.

Thus, a simulated environment is needed to perform integration tests on new tools before deployment and know exactly what to expect. The SimulatorDS suite has been developed and is now already in use for Continuous Integration and Delivery of the PANIC Alarm System at ALBA, as well as a common tool for bug replication and HMI testing.

In addition, the ongoing work on deploying replicated control systems on AWS would help to use it as a generic tool for Integration Tests for any Tango related project; the next step would integrate tango-simlib in the developed platform to provide a full-featured solution.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUDPL01

ACKNOWLEDGEMENTS

We want to thank Andrew Götz and the European Synchrotron Radiation Facility to open the possibility to collaborate in the Chaos Engineering project for Tango.

REFERENCES

[1] T.Shen *et al.*, "The Evolution Of The Simulation Environment In ALMA", in *Proc. ICALEPCS 2015*, Melbourne, Australia, Oct. 2015, paper WEPGF031

[2] J.K Munro *et al.*, "A proposed alarm handling system management plan for SNS with application to target control system", in *Proc. ICALEPCS 2007*, Trieste, Italy, paper RPPB27

[3] S. Rubio-Manrique *et al.*, "Dynamic Attributes and Other Functional Flexibilities of PyTango", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper THP079.

[4] S.Rubio-Manrique, G.Strangolino, M.Ounsy, S.Pierre-Josepth, "Validation of a MySQL Archiving for ALBA", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper WEP010.

[5] S.Rubio-Manrique *et al.*, "PANIC and the Evolution of Tango Alarm Handlers", presented at ICALEPCS'17, Barcelona, Spain, Oct.2017, paper TUBPL03.

[6] TANGO website, http://www.tango-controls.org

[7] N.Bin Ali, K.Petersen, C.Wohlin, "A Systematic Literature Review On The Industrial Use Of Software Process Simulation", *Journal of Systems and Software,* Vol. 97, Nov. 2014, pages 65-85

[8] SimulatorDS sources, https://github.com/tango-controls/simulatords

[9] Tango Python binding, https://pytango.readthedocs.io

[10] Fandango library and tools for Tango, https://github.com/tango-controls/fandango

[11] Formula recipes for SimulatorDS, https://github.com/tango-controls/SimulatorDS/blob/develop/doc/recipes.rst

[12] PANIC Website, http://www.pythonhosted.org/panic

[13] L.Pivetta *et al.*, "New developments for the HDB++ Tango Archiving System", presented at ICALEPCS'17, Barcelona, Spain, Oct.2017, paper TUPHA166.

[14] Tango Simlib, http://tango-simlib.readthedocs.io

[15] A.Banerjee, S.R.Chaudhuri, P.Patwari, L.Van Den Heever, "Data Driven Simulation Framework", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, paper WEPGF025.

[16] A.Basiri, N.Behnam, R.de Rooij, L.Hochstein, L.Kosewski, J.Reynolds, C.Rosenthal, "Chaos Engineering", *IEEE Software*, vol.33, no. 3, pp. 3541, MayJune 2016

[17] "Principles of Chaos Engineering", http://principlesofchaos.org/